Program Reasoning

12. Search Space Prioritization

Kihong Heo

KAIST

Naive Enumerative Search

- Explore the search space in increasing size of programs (i.e., Occam's razor)
- With search space pruning techniques
- But, is this enough?

iter 0	x	y				
iter 1	$x + x$	$x - x$	$x + y$	$\ldots$	$x \leq y$	$\ldots$
iter 2	$x + x + y$	$x + x - y$	$\ldots$	$\text{if } (x \leq y)\ y\ x$	$\ldots$	
iter 3	$x + x + x + y$	$\ldots$	$\text{if } (x \leq y)\ (y + x)\ x$			

Problem of Enumerative Search

- Blindly search over the large search space without any guidance
- Two major problems:
  - Scalability: #programs grows exponentially in program size
  - Quality: may overfit the I/O examples
- For example, $f(-1,0) = 0 \ \wedge \ f(0,-1) = 0$

iter 0	x	y				
iter 1	$x + x$	$x - x$	$x + y$	$\ldots$	$x \leq y$	$\ldots$
iter 2	$x + x + y$	$x + x - y$	$\ldots$	if $(x \leq y)\ y\ x$		$\ldots$
iter 3	$x + x + x + y$		$\ldots$	if $(x \leq y)\ (y + x)\ x$		

But which one is more likely to be a solution?

$x - x$ **vs.** `if` $(x \leq y)\ y\ x$

Statistical Regularities in Programs

- Programs often contain **repetitive** and **predictable** patterns

```
for (i = 0; i < 100; ??)
```

- **Statistical** program models: probabilistic distribution over programs
    - E.g., n-gram, probabilistic context-free gramma (PCFG), etc

$$\Pr(\texttt{??} \rightarrow \texttt{i++} \mid \texttt{for (i = 0; i < 100; ??)}) = 0.85$$
$$\Pr(\texttt{??} \rightarrow \texttt{i--} \mid \texttt{for (i = 0; i < 100; ??)}) = 0.01$$

- Applications: code completion, deobfuscation, program repair, etc

A Solution: Euphony*

- Enumerate programs by **likelihood**, not by program size
- Likelihood is provided by **probabilistic models** over programs
  - *"How likely is the candidate program?"*
- Try the **most likely** (highest probability) candidate first

*W. Lee, K. Heo, R. Alur, M. Naik., Accelerating Search-Based Synthesis Using Learned Probabilistic Models, PLDI, 2018

Probabilistic Language Model

- A probability distribution over sentences in a language
  - Conditional probability and the chain rule: $\Pr(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} \Pr(w_i \mid w_1, w_2, \ldots, w_{i-1})$
  - E.g., Pr(*I am a boy*) = Pr(*I*) x Pr(*am* | *I*) x Pr(*a* | *I am*) x Pr(*boy* | *I am a*)
- Learned from a large corpus of sentences

Which program is more likely?

"I am a boy"	**"You am a boy"**
92%	**1%**

Which word is more likely to be next?

"I am a KAIST ________"

"I am a KAIST stupid."

student ✕

A Naive Language Model

- $\Pr(w \mid h)$: the probability of a word w given some history h
    - E.g., $\Pr($*the* | *its water is so transparent that*$)$
- How to compute $\Pr(w \mid h)$?
- A naive idea: count

Large corpus (e.g., Web)

$$\longrightarrow \frac{C(\textit{its water is so transparent that the})}{C(\textit{its water is so transparent that})} = \Pr(\textit{the} \mid \textit{its water is so transparent that})$$

Problem: Sparsity

- What is the probability of the following sentence S?
  - S: "*KAIST's duck pond water is so transparent that the*"
  - $\mathrm{Pr}(S)$ = Pr(*KAIST's*) x Pr(*duck* | *KAIST's*) x …
    x Pr(*the* | *KAIST's duck pond water is so transparent that*)
  - C(*KAIST's duck pond water is so transparent that*) over the web?
- Language is creative: new sentences are created all the time!
- Longer sentence: lower probability

N-gram

- Idea: approximate the history by the last N words
  - Unigram, bigram, trigram, 4-grams, …
- Example: trigram
  - $\Pr(\textit{the} \mid \textit{KAIST's duck pond water is so transparent that}) \approx \Pr(\textit{the} \mid \textit{transparent that})$

Large corpus (e.g., Web)

$$\longrightarrow \frac{C(\textit{so transparent that the})}{C(\textit{so transparent that})}$$

Probabilistic Language Model for Programs

- A probability distribution over programs in a language
- Learned from a large corpus of programs (e.g., Github, Stackoverflow, etc)
- Example: N-gram, PCFG, PHOG, etc

Which program is more likely?

```
“x - x”   “if (x ≤ y) y x”
```

1% **95%**

Which word is more likely to be next?

```
“for (i = 0; i < 100; ___)”

“for (i = 0; i < 100; j++)”
```

i ×

Probabilistic Grammar (1)

- Idea: given a context, provide the probability of each production rule: $\Pr(\text{rule} \mid \text{context})$
    - Context (history): sentential form $\in (N \cup \Sigma)^*$
- Ultimately assign a probability to each program
- Example

CFG $S \rightarrow x \mid 1 \mid S + S$

Probability of “x + 1”

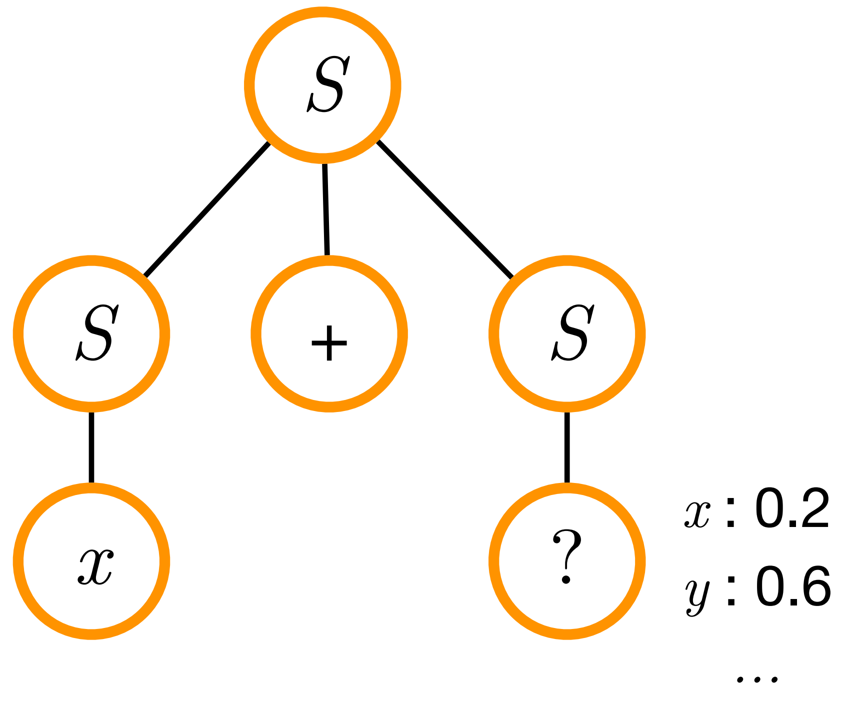
$$\underline{S \rightarrow S + S} \rightarrow x + S \rightarrow x + 1$$

$$\Pr(x + 1) = \boxed{\Pr(S \rightarrow S + S \mid S)} \times \boxed{\Pr(S \rightarrow x + S \mid S + S)} \times \boxed{\Pr(S \rightarrow x + 1 \mid x + S)}$$

Probabilistic Grammar (2)

- Learn a probabilistic model of programs from a corpus of programs
  - Human-written or auto-generated programs by other synthesizers

$$
\begin{aligned}
Pr(S \rightarrow S + S) &= 0.3 \\
Pr(S \rightarrow x \mid S + S) &= 0.8 \\
&\cdots \\
Pr(S \rightarrow x \mid x + S) &= 0.2 \\
Pr(S \rightarrow y \mid x + S) &= 0.6 \\
&\cdots
\end{aligned}
$$



Program Corpus → **Learned Probabilistic Model** → **Probability of Programs**

PCFG

- Probabilistic Context Free Grammar (PCFG)
- One of the simplest form of probabilistic language model: ignore context
- Provide a probability to each production rule

Large corpus (e.g., Github)

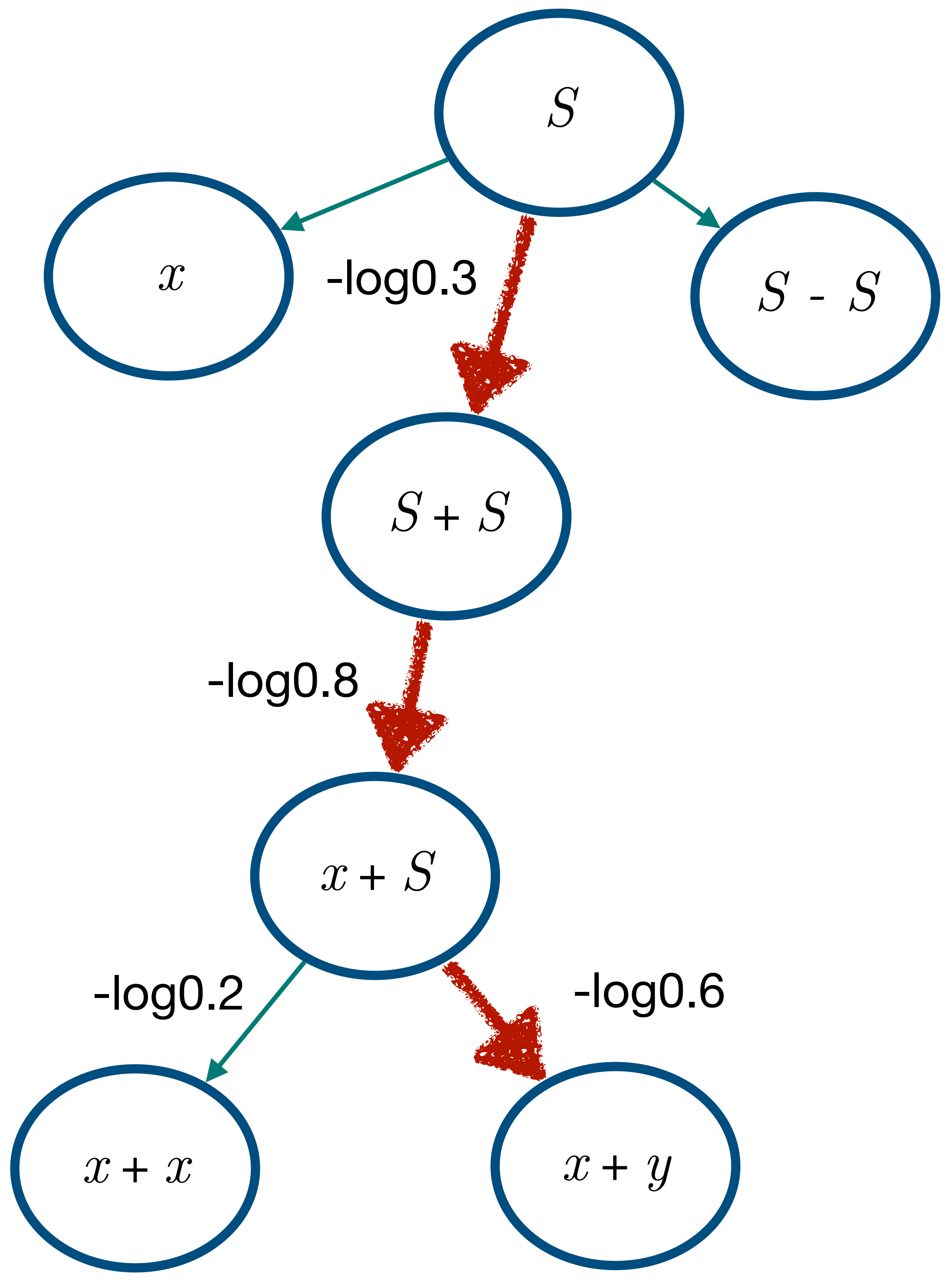
$$\frac{C(S \to t)}{C(S \to *)}$$

$A \to \beta$	P
$S \to 0$	0.1
$S \to 1$	0.2
$S \to x$	0.3
$S \to S + S$	0.3
$S \to S - S$	0.1

Guided Enumeration by Probabilistic Model

- Given a model, construct a directed graph
  - Node: sentential forms
  - Weight: negative log probability of a production rule
- Compute the shortest path
  - starting from the start symbol to the program
  - E.g., Dijkstra's, A*, etc
- Enumeration by likelihood

$\Pr(S \rightarrow x)$ = 0.3
$\Pr(S \rightarrow y)$ = 0.2
$\Pr(S \rightarrow z)$ = 0.01
$\Pr(S \rightarrow S + S)$ = 0.4
$\Pr(S \rightarrow S - S)$ = 0.09



x : 0.3 y : 0.2 $x + x$: 0.4 x 0.3 x 0.3 = 0.036 $x + y$: 0.4 x 0.3 x 0.2 = 0.024 z : 0.01

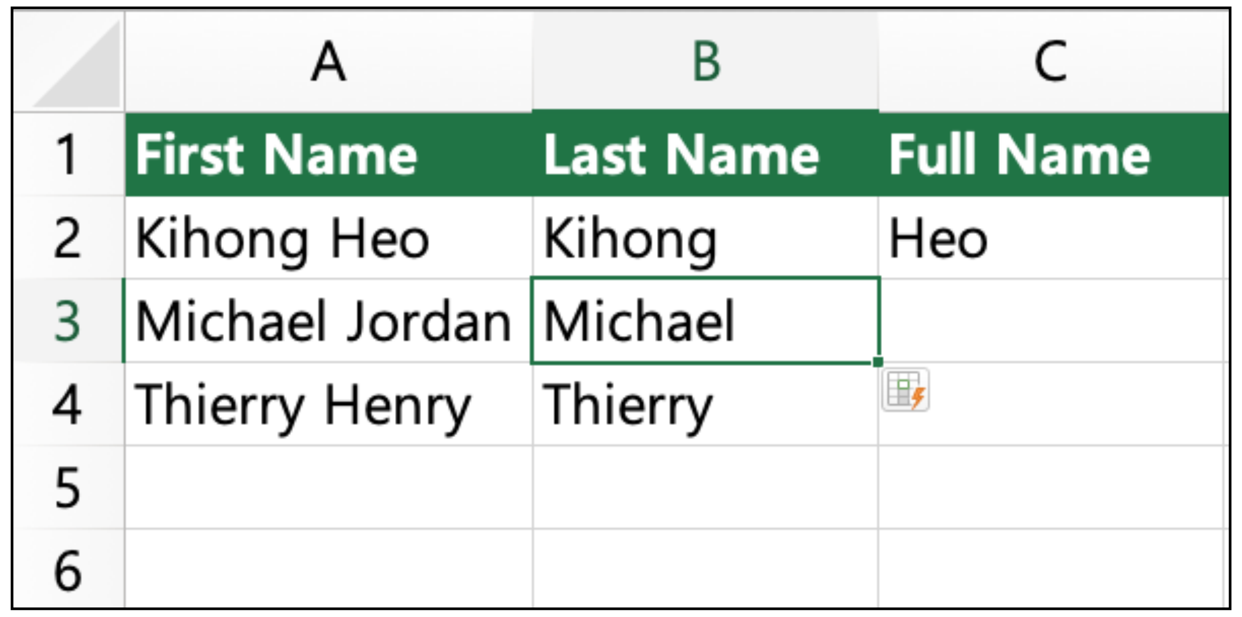
Guided Top-down Enumeration

```
top-down(G = <Σ, N, R, S>, ϕ):
  Q := {(S, 0)}
  while Q != {}:
    (p, d) := dequeue_min(Q)
    if ground(p) ∧ ϕ(p): return p
    P’ := unroll(G, p, d)
    forall p’ ∈ P’:
      if not equiv(p, p’):
        enqueue(Q, p’)


unroll(G = <Σ, N, R, S>, p, d):
  Q’ := {}
  A := left-most non-terminal in p
  forall (A → B) in R:
    p’ := p[B/A]
    Q’ := Q’ ∪ {(p’, d + w(p, p’))}
  return Q’
```

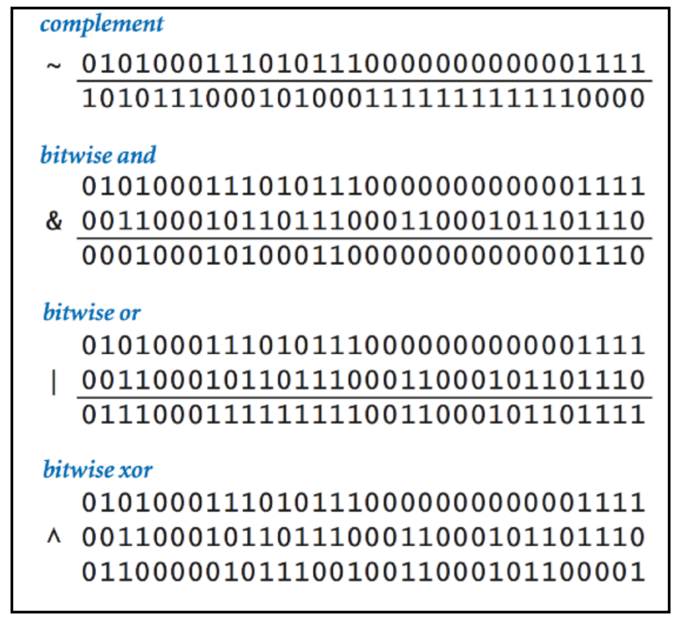
Experimental Setup

- 1167 tasks from 3 different domains



	A	B	C
1	First Name	Last Name	Full Name
2	Kihong Heo	Kihong	Heo
3	Michael Jordan	Michael	
4	Thierry Henry	Thierry	
5			
6			

STRING: End-user programming for string manipulations
(205 tasks)



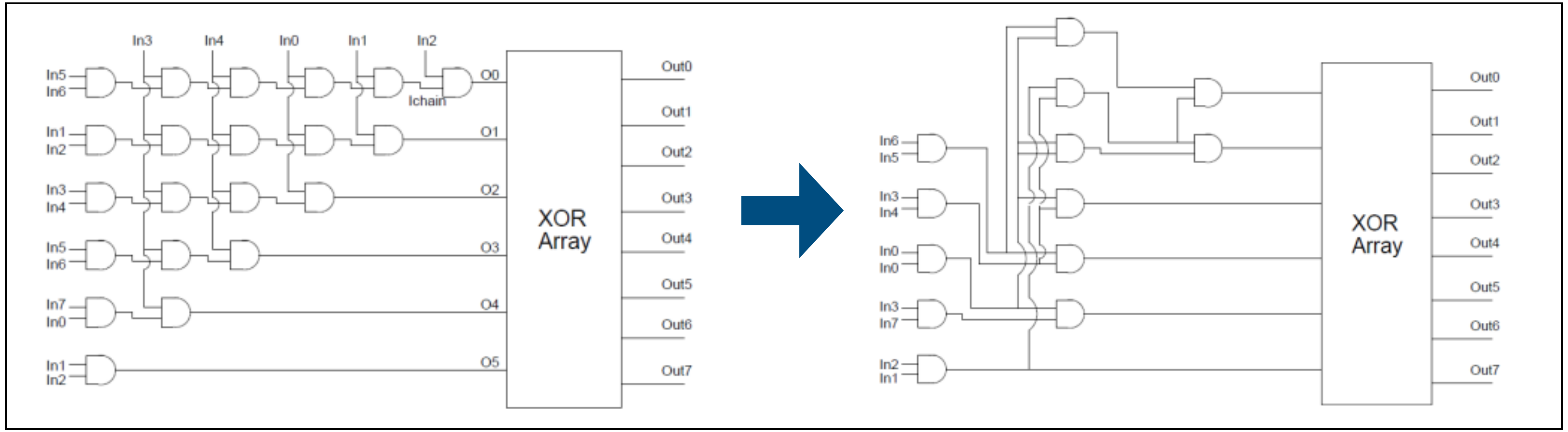
```
complement
~ 01010001110101110000000000001111
  10101110001010001111111111110000

bitwise and
  01010001110101110000000000001111
& 00110001011011100011000101101110
  00010001010001100000000000001110

bitwise or
  01010001110101110000000000001111
| 00110001011011100011000101101110
  01110001111111110011000101101111

bitwise xor
  01010001110101110000000000001111
^ 00110001011011100011000101101110
  01100000101110010011000101100001
```

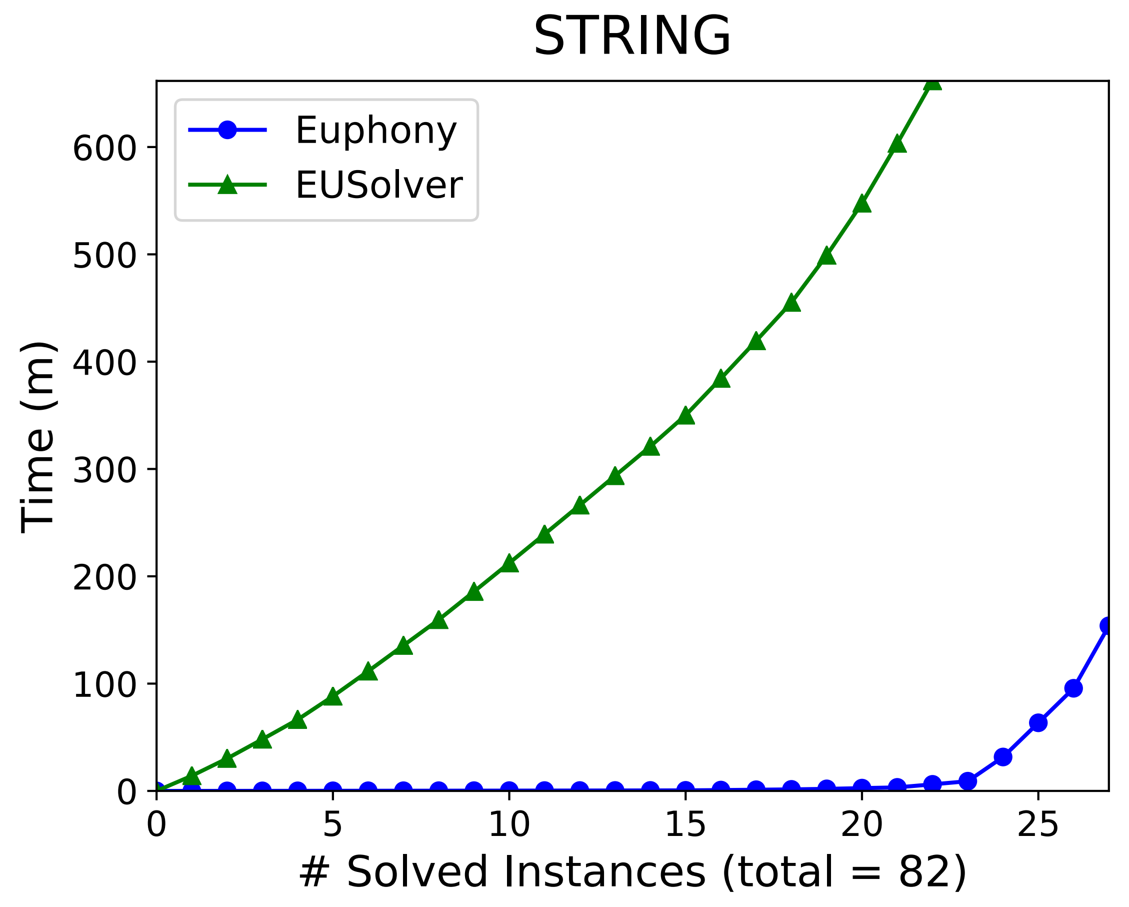
BITVEC: Efficient low-level algorithms
(750 tasks)

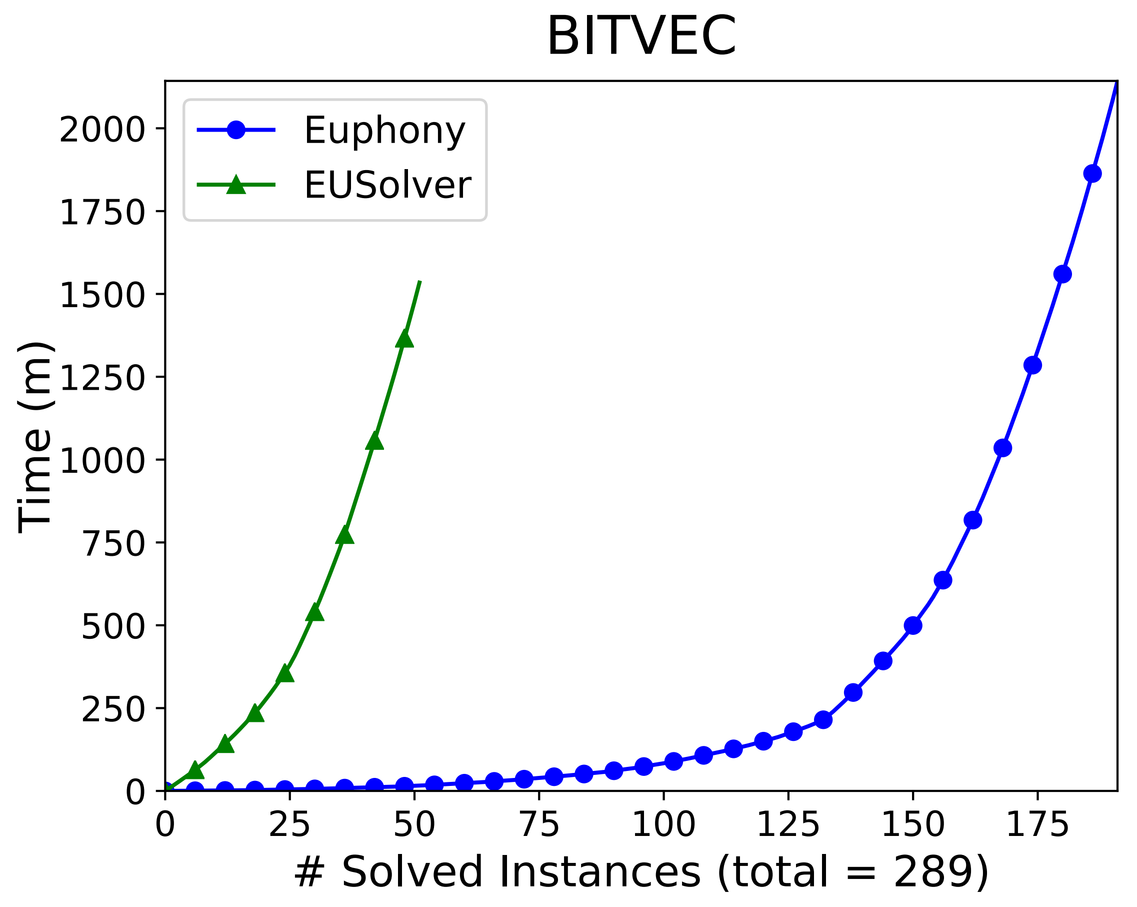


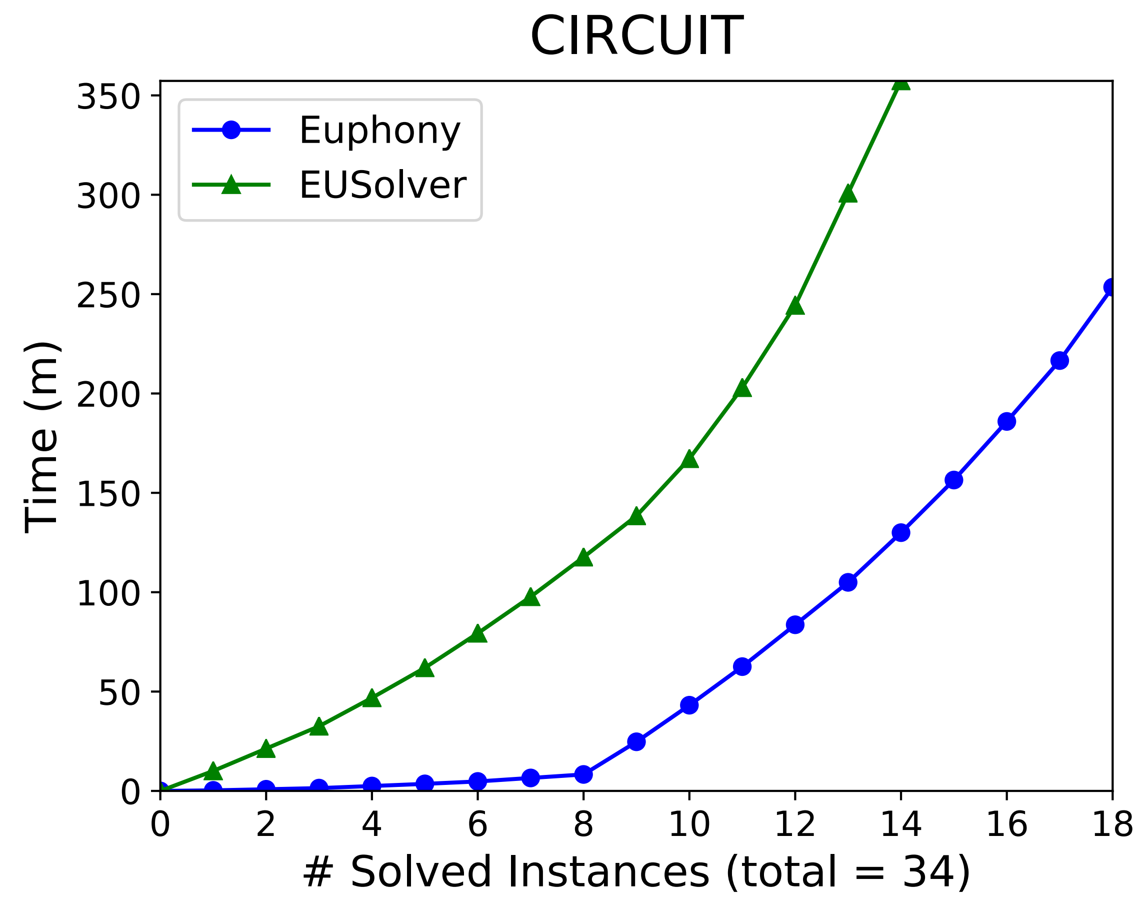
CIRCUIT: Attack-resistant crypto circuits generations
(212 tasks)

Effectiveness

- Comparison to EUSolver (a program synthesizer without prob. guidance)
  - Training: 762 tasks solved by EUSolver in 10 minutes
  - Testing: 405 (timeout: 1 hour)







Summary

- Problem: scalability and quality
- Euphony: a program synthesizer guided by a **learned probabilistic model**
  - E.g., probabilistic program model + shortest pathfinding
- Need a lot more research on efficient search
  - E.g., advanced learning techniques, static analysis, constraint solving, etc